Licence d'informatique
Module de C/Unix

Examen – Partie C

Jean-Luc LEVAIRE

Janvier 2003

Les documents de cours et TD sont autorisés.
On rendra deux copies séparées pour la partie Unix et pour la partie langage. La partie C est à rendre sur une *copie de couleur blanche.*

1 Questions de cours

Question 1

1. Quel est le contenu d'un inode ?
2. Expliciter rapidement la triple indirection des blocs de données dans un inode.

Question 2
Que signifient les déclarations suivantes ?

1. `void *(*(f[]))(int *);`
2. `char *d(int (*[])());`

2 Gestion de processus

Le but du problème est de réaliser l'ordonnancement des processus sous Unix. On rappelle qu'un processus est un programme en cours d'exécution. Unix est un système multiprocessus, et il existe donc à un instant donné plusieurs processus en cours d'exécution. Sur un système monoprocesseur, seul un unique processus se verra réellement attribuer le processeur à un instant donné: c'est le processus *actif*. L'ordonnancement consiste ainsi à définir l'ordre dans lequel les processus deviennent actifs.

L'ordonnancement le plus simple est celui qui attribue le processeur successivement à chaque processus. On attribue un quantum de temps (typiquement 1 milliseconde) à l'exécution de chaque processus, et le noyau se charge de donner le processeur aux processus l'un après l'autre. C'est l'horloge du système qui informe le noyau qu'un quantum de temps est terminé, et celui-ci passe alors au processus suivant.

Certains processus, cependant, n'utilisent pas le processeur systèmatiquement: lors d'une entrée-sortie bloquante (lecture clavier, accès à un bloc d'un disque, ...), un processus se met en attente de la fin de l'entée-sortie. Ce sont en effet les contrôleurs d'entrée-sortie qui réalisent l'entrée-sortie et non le processeur. Il faut donc supprimer pendant le temps de l'entrée-sortie le processus de l'ordonnancement sur le processeur. Un processus peut ansi prendre trois états pendant sa durèe de vie :

– prêt (Ready): le processus est prêt à utiliser le processeur;
– actif (Active): le processus est prêt et utilise actuellement le processeur;
– en attente (Waiting): le processus est en attente de fin d'entrèe-sortie. Il n'utilise donc pas le processeur pour le moment;

Au niveau de l'implémentation de la gestion des processus dans le noyau, nous utilisons:

– un tableau de processus `t_proc` dont la taille est fixée à la compilation (constante `NPROC`);

Id: xamc03.tex,v 1.3 2003/02/15 10:51:00 marquet Exp

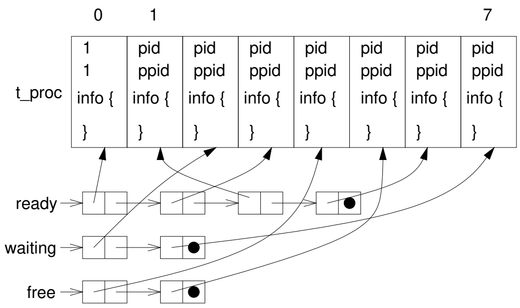

FIG. 1 – Gestion de processus. Dans cet exemple on a pris un tableau de `NPROC` = 8 processus au maximum. `ready`, `waiting` et `free` seront du type `P_list`. `t_proc` est un tableau de `NPROC` enregistrements comportant chacun un identifiant de processus `pid`, un identifiant de processus parent `ppid` et une structure `struct info`.

– une liste `ready` des processus prêts. Cette liste pointe directement dans le tableau `t_proc`. Le processus en tête de cette liste sera le processus actif;
– une liste `waiting` des processus en attente de fin d'entrée-sortie. Cette liste pointe aussi directement dans le tableau `tproc`.
– une liste `free` des cases libres de `t_proc` qui seront utilisées pour lancer de nouveaux processus;

Cette implémentation est présentée Figure 1. Dans le tableau `t_proc` sont stockés l'identifiant du processus (`pid`), l'identifiant du processus père (`ppid`) ainsi que d'autres informations qui sont regroupées dans une structure `struct info`.

Question 3
Proposer un type de données `T_proc` pour le tableau `t_proc`, et un type de données `P_list` pour les 3 listes de processus (`ready`, `waiting` et `free`).

Question 4
Écrire une procédure

```
void Init(P_list *pready, P_list *pwaiting, P_list *pfree, T_proc t_proc)
```

qui crée et initialise les trois listes pointées par `pready`, `pwaiting` et `pfree` à partir du tableau `t_proc`[1]. On créera un unique processus prêt (son nom est `init`), de `pid` 1 et de `ppid` 1, qui sera le père de tous les processus.

Question 5
Écrire une procédure

```
void Createprocess(P_list *pready, P_list *pfree, unsigned int ppid)
```

qui crée un nouveau processus. La création de processus sous Unix se fait par mécanisme de fourchette (*fork*) qui correspond en fait à un clonage de processus: le processus actif peut par ce mécanisme créer un clone de lui même, la seule différence entre les deux processus clones étant le `pid` et le `ppid`. Pour le processus ainsi créé, le `ppid` sera le `pid` du processus actif, son `pid` sera déduit d'une variable globale `unsigned int newpid` que l'on incrémente à chaque création de processus. Le processus créé sera placé dans la liste des processus prêts en tête de liste, le processus père en fin de liste.

Question 6
Écrire une procédure

[1] Notez que toutes les procèdures prennnent pour paramètres des pointeurs sur les listes

```
void Swicthcontext(P_list *pready)
```

qui modifie le processus actif courant. Cette procédure est appelée après chaque quantum de temps par le noyau pour attribuer le processeur au processus suivant le processus actif. Ce dernier sera placé à la fin de la liste des processus prêts.

Question 7
Écrire une procédure

```
void Syscall(P_list *pready, P_list *pwaiting)
```

qui correspond à un appel système réalisé par le processus actif. Cet appel système est en fait une demande d'entrée-sortie: le processus actif doit donc être placé dans la liste des processus en attente.

Question 8
Écrire une procédure

```
void Sysret(P_list *pready, P_list *pwaiting, unsigned int pid)
```

qui correspond à une terminaison d'entrée-sortie pour le processus d'identifiant `pid`. Ce processus sera ainsi déplacé de la liste des processus en attente vers la liste des processus prêts en second membre de cette liste (si c'est possible).

Question 9
Écrire une procédure

```
void Sysexit(P_list *pready, P_list *pfree)
```

qui correspond à la terminaison du processus actif. Ce dernier sera donc inséré dans la liste des cases libres pointée par `pfree`.

Fin du sujet